NOTE: Use these buttons to print or save the form. DO NOT use the browser tool bar. SAVE PRINT

LA-UR-11-02106

*Approved for public release;*
*distribution is unlimited.*

*Title:* In-situ Sampling of a Large-Scale Particle Simulation for Interactive Visualization and Analysis

*Author(s):*
Jonathan L. Woodring 209118 CCS-7
James P. Ahrens 113788 CCS-7
Jeannette A. Figg 224697 CCS-6
Joanne R. Wendelberger 110379 CCS-7
Katrin Heitmann 175878 ISR-1

*Intended for:* EuroVis 2011, May 31–June 3, Bergen, Norway

Los Alamos NATIONAL LABORATORY EST.1943



Form 836 (7/06)

SAVE PRINT CLEAR FORM

Eurographics/ IEEE-VGTC Symposium on Visualization (EuroVis 2011)
H. Hauser, H. Pfister, and J. van Wijk
(Guest Editors)

*Volume 30 (2011), Number 3*

# In-situ Sampling of a Large-Scale Particle Simulation for Interactive Visualization and Analysis

J. Woodring[1], J. Ahrens[1], J. Figg[2], J. Wendelberger[2], S. Habib[3], and K. Heitmann[4]

[1]CCS-7 Applied Computer Science Group
[2]CCS-6 Statistical Sciences Group
[3]T-2 Nuclear and Particle Physics, Astrophysics, and Cosmology Group
[4]ISR-1 Space Science and Applications Group
Los Alamos National Laboratory, U.S.A.

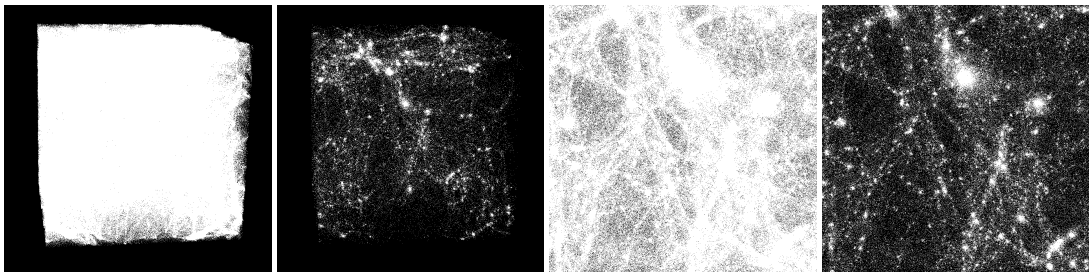

**Figure 1:** *An example of particle data from the MC$^3$ dark matter simulation. The images show the comparison between full resolution data and statistically-based level-of-detail data samples generated via in-situ sampling.*

**Abstract**

*We describe a simulation-time random sampling of a large-scale particle simulation, the RoadRunner Universe MC$^3$ cosmological simulation, for interactive post-analysis and visualization. Simulation data generation rates will continue to be far greater than storage bandwidth rates by many orders of magnitude. This implies that only a very small fraction of data generated by a simulation can ever be stored and subsequently post-analyzed. The limiting factors in this situation are similar to the problem in many population surveys: there aren't enough human resources to query a large population. To cope with the lack of resources, statistical sampling techniques are used to create a representative data set of a large population. Following this analogy, we propose to store a simulation-time random sampling of the particle data for post-analysis, with level-of-detail organization, to cope with the bottlenecks. A sample is stored directly from the simulation in a level-of-detail format for post-visualization and analysis, which amortizes the cost of post-processing and reduces workflow time. Additionally by sampling during the simulation, we are able to analyze the entire particle population to record full population statistics and quantify sample error.*

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications— H.3.m [Information Storage and Retrieval]: Miscellaneous—

## 1. Introduction

Modern scientific discovery greatly depends upon computer modeling and interactive exploratory visualization has proved to be essential for scientific discovery and analysis [AHP*10]. For scientific simulations, current leadership supercomputing is in the petascale range ($10^{15}$ flops). It is expected in the next decade, supercomputers will progress into the exascale range ($10^{18}$ flops). The amount of data gen-

erated through simulations on these machines is a significant concern to the scientific simulation community, as the generated sizes will surpass the capability to interactively explore and analyze the data [JR07].

Primarily, this is because storage read and write bandwidth on leadership supercomputing has not kept pace with computational speed [AHL*10], therefore it is not possible to store all generated data for post-analysis. This problem is generally recognized as data intensive computing or data intensive supercomputing (DISC). For example, the RoadRunner Universe MC$^3$ [HPL*09] is a large $N$-body cosmology simulation of dark matter physics. An MC$^3$ time step of $4000^3$ (64 billion) particles with 36 bytes per particle takes 2.3 TB per time slice. A Panasas parallel filesystem [pan] connected to Los Alamos National Laboratory (LANL) simulation supercomputers can operate at 10 GB/s. At an optimistic peak, it takes 4 minutes to move the $4000^3$ particle data to or from storage in parallel, which takes a large hardware infrastructure investment for the bandwidth. Under operating conditions, it can take longer than 4 minutes to write the data under heavy load. Very few time steps, compared to the entire data generated over time, are written to disk due to the I/O to compute imbalance and storage size limitations [JR07, SBH*10].

The contribution of this paper is a study of an *in-situ* (analysis or visualization as part of the simulation) statistical sampling method for the MC$^3$ simulation, which stores level-of-detail sample data for later interactive post-analysis and visualization, shown in Figure 2. Our first goal is to circumvent the storage bottlenecks which hinder simulations from storing data for post-analysis. We improve performance by explicitly reducing the amount of stored data per time slice via *in-situ* analysis and sampling. Our second goal is to provide support for interactive scientific analysis and visualization by providing efficient and accurate "raw sample data." The stored sample data can be used directly in scientific analysis and we quantify sample accuracy through measuring the statistics of the original data *in-situ* compared to sample data. Furthermore, we directly encode sample data into a level-of-detail format to amortize the cost of post-processing and allowing for immediate interactive visualization.

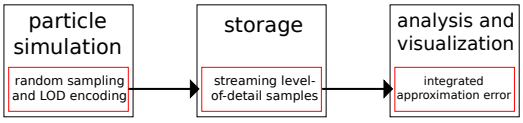


**Figure 2:** *The end-to-end pipeline from data generation to exploratory post-analysis of large-scale particle data.*

## 2. Related Work

Simulations are limited by storage bandwidth and capacity and can only store a small fraction of data compared to the entirety of the generated data (estimated to be under 10%) [AHL*10, JR07, SBH*10]. The standard methods to cope with the lack of I/O bandwidth at simulation run-time are to store very few time steps, drop data dimensions (variables), drop precision (converting doubles to floats), and/or use *in-situ* visualization [MFMG10, TCM10, TYR*06, YWG*10]. When performing *in-situ* visualization, there is a trade-off of what data to save for later analysis and what operations can be performed afterward [TCM10]. We use *in-situ* analysis to reduce the storage bandwidth and store raw data to provide exploratory visualization (most analysis operations) in post-analysis.

Other techniques have reduced data sizes in post-processing through compression and quantization [BR09, CMNR07, EGM05, LI06, PF01, WGLS05]. We introduce one more method for reducing the size of data for scientific visualization via statistical sampling [Coc77, Loh10, Vit87]. Random sampling of particles allows us to spread the information loss [CJ10] across spatial dimensions. The benefit of sampling at simulation-time is that we are able to record the *approximation error* [LHJ03, WGS07] (how accurately sample data represents the original data). With this information, we visually highlight the particle samples with high approximation error [JS03, PKRJ10].

Standard practice in large-scale visualization is utilization of massively parallel visualization methods [CPA*10]. Alternatively, there are methods to post-process data to support interactive visualization via streaming, level-of-detail (LOD), and prioritization [AWD*09, CMNR07, FSW09, HE03, LHJ99, PF01, WGLS05]. LOD visualization is a useful technique to allow large data to scale under the run-time bottlenecks in storage, networks, and displays [Dee98]. Therefore, the sample data we acquire is stored in an LOD structure that is read by a multi-resolution visualization system [AWD*09].

Though, the practice of post-processing [CMNR07, EGM05, FSW09, HE03, PF01, Vit87, WGLS05] for interactive visualization will not scale due to the storage bottleneck. We believe that post-processing is already inaccessible to scientists and the scientific workflow. The authors of [FSW09] cite that it only took 5 hours to process their data for visualization but that is 5 hours of scientist time. Additionally, only a very small amount of data can be stored compared to the total amount of generated data. Post-processing cannot recover the lost data that is never stored. Therefore, we encode the original data directly from the simulation into an LOD format, amortizing the cost of LOD encoding, and removing a step in the workflow by eliminating one read and write of the data.

## 3. Stratified Random Sampling of Particle Data

The high-level overview of the pipeline from beginning to end is encapsulated in Figure 2. In the first step, we generate samples of MC$^3$ simulation data per time slice and store it

directly into a level-of-detail (LOD) format. In the second step, the cosmologists are able to analyze the LOD sample data in a multi-resolution visualization tool [AWD*09]. In this section, we describe the general algorithm for creating a sample of particle data. In Section 4, we describe how to integrate the LOD storage mechanism with the sampling algorithm. In Sections 5, 6, and 7, we describe the visualization and analysis of the simulation with sample data. Section 8 describes the performance characteristics of our method and we conclude with Section 9.

We generate an *in-situ* random sample from the MC$^3$ particle data during simulation by sampling from spatially contiguous blocks of particles. Each block or stratum has an equal number of samples that are allocated proportional to the number of particles in a block. This sampling strategy is a self-weighting *stratified random sampling* (which we abbreviate to StRS, since SRS refers to simple random sampling) [Coc77, Loh10]: each stratum has the same population size, the number of random sample points taken from each stratum is proportional, and the inclusion probability of every particle in the final sample is equal.

There are several reasons that we utilize StRS for particle sampling: From a statistical standpoint, stratified random sampling helps to acquire a good sample [Coc77, Loh10]. The estimator for the mean is unbiased and has a lower variance than pure random sampling. From a systems perspective, most simulations run where the data is simulated in a spatially contiguous region per processing element. Therefore, the data is naturally separated into spatial strata. In the MC$^3$ cosmological simulation, it simulates the dark matter particles in this manner. Additionally by separating the particle data into strata for sampling, we can easily modify the sampling to generate blocked particle data for a streaming level-of-detail (LOD) visualization system.

### 3.1. Implementing StRS with Median Value kd-tree Sorting

We assume that the particle simulation is pre-partitioned per processor into large spatial blocks, as is the case with the MC$^3$ simulation. To acquire a sample size of $s$ from $N$ particles, we allocate a proportional number of samples per processor block. Assuming there are are $n_i$ particles on processor $p$, block $n_i$ is allocated $n_i \cdot s/N$ random samples. The complete sample is the union of all of the random samples taken from every processor.

We utilize a median-split kd-tree per processor to recursively divide the particle data into smaller sampling strata. The kd-tree sorts the particle data by spatial axes to divide the data by the median value per axis into equal population partitions (strata). The dividing the space into many strata ensures a more even spread of random samples across space; it is somewhat analogous to signal anti-aliasing by putting more samples in high frequency areas.

Our StRS implementation takes the same amount of time as constructing a parallel kd-tree: $O((N/p)\,log\,(N/p))$, where $N$ is the total number of particles and $p$ is the number of processors. Assuming that simulation particle data is stored in a linear array in memory and the array can be sorted (such as with MC$^3$, the particle order does not matter), the kd-tree can be constructed in place with no extra memory, using median sort (STL nth_element). In other simulations that do not allow reordering of the particle array, we can store an array of indices to restore the previous order of the data after kd-tree sorting or copy the particle data if there is enough memory.

Figure 3 graphically shows how particle data on a processor is sorted into kd-tree leaves (spatial strata) and one random sample is taken per stratum. Given a spatial block of particles per processor, we recusively split the data per axis, by median axis value, to create a spatial decomposition of the data. This sorts the data into spatial bins with an equal number of particles per bin. At the leaf nodes of the kd-tree, one random sample is taken per leaf or stratum. The samples in aggregate across all of the leaves and all of the processors is the stratified random sample. A more detailed account of the algorithm with psuedo-code can be found in the supplemental material.

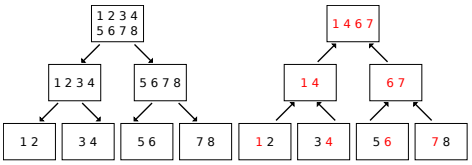


**Figure 3:** *The left tree shows 1D particle data being sorted into a kd-tree. The right tree shows the generation of a four particle random sample of the data. One random sample (red numbers) is taken per stratum (leaf nodes) and joined into one aggregate sample.*

## 4. In-situ Sampling with Level-of-Detail Construction

We extend the kd-tree stratified random sampling (StRS) algorithm to directly generate level-of-detail (LOD) output data. The sampling and LOD generation is performed such that the sampling and simulation storage methods are integrated into one algorithm. Therefore, the total time is still $O((N/p)\,log\,(N/p))$.

The output storage format is similar to other tree based LOD structures, with spatial blocks of sample particles laid out linearly on storage [AWD*09, CMNR07, FSW09, HE03, PF01, WGLS05]. In each level of the hierarchy, we store spatially contiguous blocks of particles, each of which represent a StRS of a spatial region of the simulation. Each block in the LOD hierarchy has a number of particles equal to the sample size $s$ and the number of blocks per level is $2^v$ where $v$ is the level ($v \geq 0$). For simplicity, we assume the number of processors are a power of 2 and the sample size is a power of 2.

Figure 4 shows a diagram representation for creating the LOD hierarchy per processor with StRS. An LOD block in the hierarchy represents a stratified random sample of all of the particles in a spatial region for a particular level-of-detail. Samples from higher resolution LOD blocks are propagated up the hierarchy to lower resolution LOD blocks. We ensure that each strata, in a lower resolution LOD block, is only sampled once (ensuring it still is a StRS) by taking one random sample for every $k$ particles, where $k$ represents the number of high resolution strata merged together to create a low resolution block.

Figure 5 shows the completion (reduction) of the LOD hierarchy, in parallel, from per processor samples. To complete the LOD hierarchy, the LOD blocks created per processor are random sampled by using the same merging criteria as before. A more detailed account of the algorithm with psuedo-code can be found in the supplemental material.

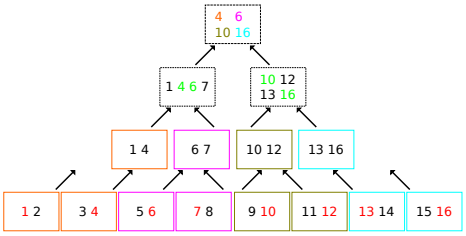


**Figure 4:** *The creation of a two-level LOD hierarchy with sample size of four per block (black nodes at top), for 1D particle data on one processor. At the bottom of the kd-tree are the finest grain strata and the red numbers are the random samples for the bottom of the LOD hierarchy. The green numbers are samples propagated to a lower resolution level (one random sample per k samples, this case 2). We ensure that the lowest resolution sample (top of the tree) only has one sample per stratum (color coded nodes and numbers).*

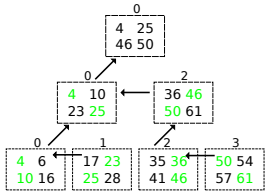


**Figure 5:** *Completion (reduction) of an LOD hierarchy with sample size of four per block, for 1D particle data across four processors. The green numbers are the random samples propagated to lower resolution levels (one random sample per k samples, this case 2).*

### 4.1. LOD Storage Size

Selected levels in the LOD hierarchy are written to storage. In our implementation, we duplicate particles occurring in each block at different resolutions, i.e., if particle $A$ exists in $n$ levels of the hierarchy it will be replicated $n$ times. In Section 4.1.1, we provide the rationale for particle duplication in the LOD storage. Given our storage implementation, the number of particles stored per LOD level is $2^v \cdot s$ where $v$ is a level in the LOD hierarchy ($v \geq 0$) and $s$ is the sample size ($1 \leq s \leq N/2$). Given $V$ ($V \subset Natural$), a set of LOD levels to store, the total number of particles written to storage will be $\sum_{v \in V} 2^v \cdot s$. This sum will be less than or equal to the total number of particles $N$, assuming that $\forall v \in V : 2^v \cdot s \leq N/2$ (the highest resolution LOD level is half the original data).

This is because the highest possible resolution of a level, which is not the full resolution data, is at most one random sample of every two particles. The next possible highest resolution data will be $1/4$ of the original data, the next will be $1/8$, etc. (lower resolution levels are at a maximum a $N/2$ sample size). Given this geometric series and Equation 1, the LOD tree will at most be $N$ of the original data, where $degree = 2$ (the kd-tree degree). This is because the size of the data is $N/(degree-1) = N/(2-1) = N/1 = N$.

$$\begin{aligned} \sum_{k=1}^{depth} \frac{N}{degree^k} &= N \frac{degree^{depth}-1}{degree^{depth}(degree-1)} \\ \sum_{k=1}^{\infty} \frac{N}{degree^k} &= \frac{N}{degree-1} \end{aligned} \tag{1}$$

$N$ or $2N$ (adding the full resolution data per time slice) is too large for our use case because of storage bandwidth limitations, unless we are only concerned with amortizing the cost of building the LOD structure and removing preprocessing time. We are able to scale the size of the stored data and significantly reduce the amount of written data to fit within the storage bandwidth. We reduce the size by storing only a few selected levels of the LOD hierarchy, limiting the depth of LOD hierarchy, and/or not storing the full resolution data.

For example if we store every third level in the kd-tree, starting with the level that is $1/8$ of the data, at most the stored data will be $N/7$ of the original data set. Every level will be $1/8$ of the previous level as this follows from the construction algorithm if every third level is stored. Thus, $N/8 = N/degree$, $degree = 8$ and $N/(degree-1) = N/(8-1) = N/7$. In another example, if we only store data starting from root (zeroth) and third levels of the kd-tree, the data size will be $2^0 * s + 2^3 * s = 9 * s$, where $s$ is the sample size, which can be significantly less than $N$ depending on the size of $s$.

#### 4.1.1. Particle Duplication Discussion

The primary reason for particle duplication is to optimize reads for any arbitrary resolution. For scientific analysis tools, all of the particles for any particular resolution can be read by one linear read of a level-of-detail, without skipping across the file. For LOD visualization, a block of particles can be frustum culled at run-time to avoid reading the data

from disk. If we do not duplicate particles across levels-of-detail, there are two obvious choices of block organization on disk that will lead to frequent striding (skipping) through an LOD particle file. The first option would store the particles in contiguous high resolution blocks. To read the lowest resolution block in this case, we would need to read a portion of all of high resolution blocks (the particles in a level-of-detail are not contiguous). Conversely, the second option would store the particle data in a progressive format such that a high resolution block is read by accumulating a portion of every level-of-detail, i.e., the particles in a level are contiguous but every block is scattered across the file.

In our empirical studies when block data is spread across the LOD file, a multiple block read becomes the worst case in many stride sizes, read sizes, and block sizes, which is equivalent to reading the entire file. While this isn't an issue for small LOD hierarchies, it is an issue when the LOD is multiple gigabytes or terabytes making many worst-case reads take minutes. With particle duplication, we eliminate fine grain striding and any read strides become block size or greater. Therefore, a multi-block read for a fixed resolution only has to touch one linear segment of the LOD file corresponding to one level-of-detail rather than seeking through entire file. See Figure 14 for an impact of block striding within a level. We additionally use the particle duplication between levels to simulate fine-grain levels in Section 4.2.1.

At this time, particle duplication is not a definitive solution and it requires further exploration due to the potential write time improvement. In general, our algorithm does not require duplication of particles, as it is primarily an LOD read optimization and can be easily modified to remove particle duplication. Secondly, we are challenging the validity of reading progressively stored data, therefore our claims are going to be controversial. Finally, we could potentially eliminate particle duplication via parallel file system striping, solid state storage, static file optimization [HPS09], and/or using very large block sizes with few levels-of-detail to optimize stride sizes. Removing particle duplication would be beneficial, as it would halve the storage size and write time of the two largest LOD hierarchy cases ($2N$ becomes $N$ and $N$ becomes $N/2$) by reducing the size and write time of any LOD storage to the highest resolution.

### 4.2. LOD Visual Continuity

In the sampling algorithm, every level-of-detail (LOD) and block is a stratified random sample (StRS) of the original data, but each level and block is also a subset of higher resolution levels and blocks, i.e., levels are not completely independent random samples. We preserve this subsetting property ensure there is visual continuity while changing between LOD levels: particles do not pop in or out ("sparkle") on resolution changes. By transitivity, if $A \subset B$ and $B \subset C$, then $A \subset C$, which means that by moving from any LOD level $A$ to $C$, $A$ will always be visually depicted in $C$. Figure 6 shows an abstract depiction of the visual continuity between LOD levels.

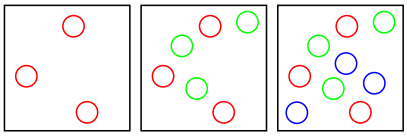

**Figure 6:** *An abstract depiction of LOD particle data under increasing resolution with visual continuity. The particles in the lower resolution data are always present in the higher resolution data.*

This follows from the sampling algorithm such that each block in the LOD structure is the combination (reduction) of two or more higher resolution blocks. We create a lower resolution block through StRS on the higher resolution blocks (randomly picking a particle from every $k$ particles, or one random sample from every stratum). Through the recursive algorithm, a block at level $L$ is the combination of two or more blocks from level $M$ where $M > L$. A lower resolution block is created by randomly taking one out of every $2^{M-L}$ particles (the ordered list of particles by kd-tree sorting), as seen in Figure 4 and 5.

#### 4.2.1. Smooth Visual Continuity

We have added an additional optimization to the LOD visualization for smooth visual continuity between displayed levels. If we directly switch resolutions between stored levels, the new amount of particles would, at a minimum, always be half or double the number of visible particles creating a large visual discontinuity. We remove the popping by smoothing the number of visible particles over time.

In the sampling algorithm and particle duplication, a low-resolution block is created from the combined particles from high-resolution blocks. Particles to create a low-resolution block from high-resolution blocks are divided into two partitions: particles that are selected (sampled) to go to the lower resolution parent block and particles that are not (particles that only exist in a higher resolutions). The higher resolution particles, which are not filtered up the hierarchy to a lower resolution, are written to storage in a randomly permuted order.

During visualization and analysis, an LOD level can be partially rendered to simulate finer-grain LOD levels. This is done by rendering all of the particles that are duplicated in a lower resolution level and incrementally adding or removing particles from the permuted list of particles that only exist in higher resolutions. Figure 7 shows the particle organization of blocks on storage and the particle duplication between low-resolution and high-resolution blocks.

## 5. Approximation Error Measurement

In the sampling algorithm, particles are sorted into spatial blocks or strata. One random particle is acquired per stratum.

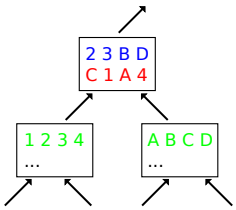


**Figure 7:** *An intermediate resolution between two stored levels can be created by a partial block. The top node is a low-resolution block created from the green particles sampled from two high-resolution blocks. The blue particles are particles duplicated in a lower resolution block (not shown). The red particles are the particles that are only duplicated in higher resolution blocks. To simulate a finer-grain LOD level, we render all of the blue particles and incrementally add red particles.*

The aggregate sample is representative of the whole population and samples in a localized region may be representative of the local area. While the aggregate sample and sample estimators may be a good representative of the whole data, the problem is that a single sample can be a poor representative a localized region.

There is a loss of transmitted information from the simulation to the cosmologist [CJ10] by sampling. The stratified random sampling (StRS) introduces *approximation error* into the representative data. We define *approximation error* to be how accurately our sample data represents the original data. This has always been the case in large-scale simulations, as the storage bandwidth and capacity are not large enough to accommodate and transmit all of the information from the simulation to storage, usually losing precision or time accuracy. [AHL*10, JR07].

The way we are able to overcome some of the information loss is that we measure and quantify information about the original data, during the simulation, to augment the sample data. The information is used to quantify the quality of the sample data, measure the amount of introduced error compared to the original data, and visualize this information. This provides a data quality assurance as we directly query the generated source data at run-time.

The measures that we utilize for $MC^3$ are the local statistical properties on the particle fields (variables). The particle data contains position (x, y, z), velocity (vx, vy, vz), and mass. We measure the population mean and variance of the original data within each stratum. The mean and variance are important measures because of their relationship to signal-to-noise ratio and the coefficient of variation [Coc77, Loh10]. These statistical properties are assigned to the sample particle taken from each strata to represent the local statistical properties of the original data. In Section 6.1, we discuss the visualization of the error metrics.

The statistical metrics also give us a mechanism to sample more in high variance areas or where the local sample mean does not mirror the local population mean, assuming that the stored LOD hierarchy will not store highest resolution data. We have not fully tested this particular feature yet, and it requires future testing because it allows the kd-tree to grow unevenly and uses more storage bandwidth than a static amount.

## 6. Streaming LOD Analysis and Visualization

To visualize our level-of-detail (LOD) $MC^3$ data, we utilize a multi-resolution visualization and analysis extension to ParaView and VTK [AWD*09]. We acknowledge that our particle rendering may be considered primitive compared to other visualization techniques [FSW09, HE03]. Nothing precludes us from integrating a more sophisticated rendering method in the future.

Our primary goal was an end-to-end data management system for direct scientific analysis of $MC^3$ sample data. The reason for this focus is that while qualitative interactive visualization is an important tool for insight, the scientific workflow and discovery process is completed with quantitative scientific analysis [AHP*10]. This has been one of our main motivating goals for storing the $MC^3$ data in a raw form, rather than being a "visualization optimized" form: the data can be utilized directly in scientific visualization and analysis tools without data transformation. For example, we can reuse existing scientific analysis tools, including the cosmologists' own tools, such as as histograms, thresholding, or specific cosmology filters developed for $MC^3$ like friends-of-friends halo finding [WHA*10] in ParaView and VTK.

### 6.1. Visualization of Local Approximation Error

By recording approximation error, each sample particle has the local statistical properties of the original data. There are two main ways that we have visually utilized this information: highlighting areas that have high variance in the original data and highlighting areas where the local sample mean differs from the local population mean. In both these cases, the visualizations are a mechanism to direct the cosmologist to parts of the data that may be under-represented and not accurate.

For the first case, we highlight areas that have high variance in low resolution data. This visualization directs the cosmologist to potential interesting areas of the data set. When viewing low-resolution LOD data, the cosmologist is prompted zoom into the area by highlighted particles. This is because the highlighting indicates there is a large spread of values in higher resolution data. Figure 8 shows the identification of a halo (cluster of dark matter particles) in a low density region, at low resolution, due to velocity variance color highlighting.

In the second example, we compare the local sample mean

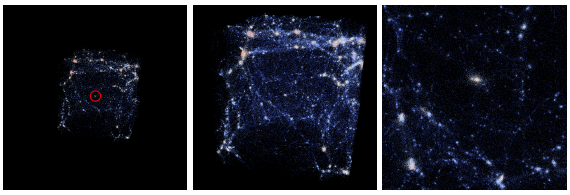

**Figure 8:** *A halo in a low density region is visually located in a low resolution sample by velocity variance color highlighting. LOD resolution increases left to right with zooming.*

to the local population mean, which has been previously calculated *in-situ*. If the two means differ significantly, we highlight those areas to show that the sample estimation in a local region may not be a good representative. As before, the cosmologist should zoom into a higher resolution sample before doing any calculations on the data in that localized area.

## 7. Analysis and Comparison of Sample Data

We compare the accuracy of a low resolution MC$^3$ sample in statistical measures and data value variation across space in the velocity field. Figure 9 shows several histograms of the velocity components (top row) and the velocity component variation across space (bottom row). The red line is a 32768 sample (0.19%) of a 256$^3$ particle data and the black line is the value of the original data. As we can see in the histograms, the sample data almost exactly represents the histogram of the velocity components (the two curves overlap each other). The velocity, as it varies across space, also nearly matches the original data except in the velocity $x$ component, where it varies slightly more, compared to velocity $y$ and $z$.

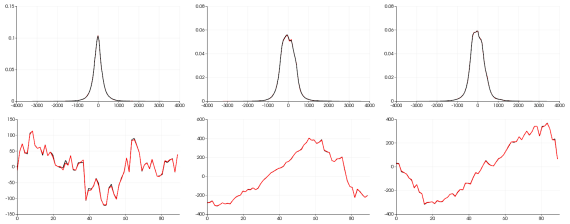

**Figure 9:** *Graphs showing a 32768 sample (red curve) of a 256$^3$ MC$^3$ particle data set (black curve). Both curves exist in all graphs; they occlude each other. Top row is the histogram of the velocity components (vx, vy, vz, left to right). Bottom row is the average value of the velocity component across space (vx in x, vy in y, vz in z, left to right). In both cases, the curves overlap each other showing that the sample is a good approximation (the black occludes the red curve in the top graphs, and vice-versa on the bottom graphs).*

Secondly, we measure the spatial sampling accuracy by using spatial feature finding and statistical measures on that feature. In cosmological science, halos, a clustering of dark matter particles, are an important feature in dark matter simulations. An important measure is the histogram of halo masses, called the "halo mass function". In Figure 10, we show the comparison of running a halo finding algorithm and the subsequent halo mass function on different sample sizes of a 256$^3$ particle data set. We scale the halo finding parameters based on the sample size. The smallest sample size, 0.19% of the data, is not able to replicate the mass function. On the other hand, the samples of 1.6% and 12.5% are able to approximate the full resolution mass function.

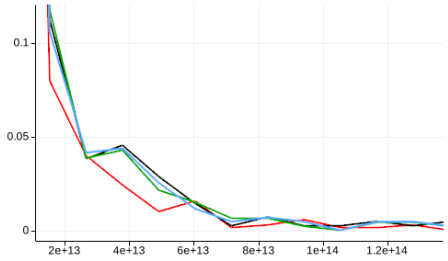


**Figure 10:** *The halo mass function for different sample sizes of 256$^3$ particles. The black curve is the original data. The red, green, and blue curves are .19%, 1.6%, and 12.5% samples, respectively.*

### 7.1. Discussion of General Applicability

Our sampling method is generally applicable to different simulations with statistical estimators, variance, and other statistical analyses. General scientific analysis already commonly uses sampling and statistical analysis. This work brings some of that existing knowledge into computational simulation and visualization. The number of samples that are required for error constraints and guarantees is completely data and variability dependent. A simple analogy is that a constant field only requires one point to represent the entire data set, while a complex function needs many more sample points to have low variability estimators.

At this point for more complex analysis such as halo finding, the applicability of sampling is data and analysis dependent, on a case by case basis. For example, we tested whether MC$^3$ sampled data could be used for halo finding. We determined that the input parameters needed to be scaled by the size of the sample and found the appropriate sample sizes through empirical testing. For other data sets and analysis, the same process would need to be performed to determine applicability, analysis parameters, and sample sizes. Though, it is a worthwhile point of future research to try to determine if there is a method for pre-determining sample sizes for guaranteed error constraints and whether that can be can be generalized to error bounds for function reconstruction and scientific analysis.

## 8. Resource Timings

A RoadRunner Universe MC$^3$ 2048$^3$ particle run generates 8 billion particles per time slice. A time slice is 309 GB total at

36 bytes per particle. In a 512-way run, the per process files are approximately 604 MB each. The data is stored from the simulation on a Panasas file system [pan] connected to Cerrillos, a hybrid 4 x AMD/4 x Cell supercomputer with 16 GB each, featuring IBM QS22 and LS11 blades similar to RoadRunner. The nodes are connected via 4 x DDR Voltaire Infiniband in a fat-tree topology.

In our implementation, only a block size (sample size) extra memory is required as the particles can be kd-tree sorted in place with STL nth_element. On the processor reduction phase, we use an optimization such that the particles are sampled per processor before merging (reduction). Then, the particles are gathered to the write processor for LOD block storage. This requires only an extra block size of memory to gather a sample block across $2^d$ processors, where $d$ is the largest level gap in the across-processor gather hierarchy.

In the following, we time the creation for sampling and storage of an LOD structure of $2048^3$ $MC^3$ particles. The data sizes are from using our particle duplication in Section 4.1. Table 1 shows the different configurations using a 32768 sample size per block (~1.1 MB per block). The first row is the original full resolution data storage write method from $MC^3$. The second row is data size for writing the full resolution complete LOD hierarchy. The third row is the data size for writing the complete LOD hierarchy without the full resolution data (half resolution maximum). Fourth through eighth rows are the data size for writing the LOD hierarchy with selected levels (skipping $x$ number of levels in the output). We can see that the largest impact for an LOD data size is dependent on the highest resolution level.

| write config | size | max res | total data |
|---|---|---|---|
| original write | 309 GB | N | N |
| full res LOD | 618 GB | N | 2N |
| half res LOD | 309 GB | N/2 | N |
| skip 1 level (1/4 max) | 103 GB | N/4 | ~.333N |
| skip 2 levels (1/8 max) | 44 GB | N/8 | ~.142N |
| skip 3 levels (1/4 max) | 82 GB | N/4 | ~.265N |
| skip 4 levels (1/8 max) | 39 GB | N/8 | ~.126N |
| skip 5 levels (1/64 max) | 4.9 GB | N/64 | ~.016N |

**Table 1:** *Different data sizes for different LOD configurations with a 32768 sample size per block with particle duplication per level (see Section 4.1). The top row is the normal full resolution write of the "cosmo" format without sampling and LOD storage.*

Table 2 shows the amount of time savings for the cosmologist's workflow between simulation and interactive analysis. This is assuming we differ from a traditional visualization workflow, where the data is processed before visualization (one extra read and write of the data is performed). It takes 210s to write the data in its original full resolution "cosmo" format from $MC^3$ and 153s to read that same data, both 512 way from the Panasas. Therefore, we have a savings of 364s (~6 minutes) per time slice between simulation and analysis by performing the LOD encoding at simulation time. This is because we remove one extra read and write of the data in the workflow by elimination of data post-processing. Also with our method, the sample data is immediately ready for interactive visualization after it is written to disk, allowing for simulation monitoring.

| original write | full read | workflow savings |
|---|---|---|
| 210s / slice | 153s / slice | 364s / slice |

**Table 2:** *Original full resolution $MC^3$ data read and write timings. "Workflow savings" is the time savings between simulation and post-analysis per time slice through* in-situ *processing.*

In Figure 11, we show the actual time taken to write the data in different LOD configurations. The first bar shows the original write time per time slice. The next two bars show that encoding a full or half resolution LOD structure, with particle duplication, level does not save any simulation time. Though, there still is a total end-to-end time savings if you consider the analysis lag time of 364s / slice (Table 2). Sampling and writing the data into an LOD structure (3rd bar) has an overhead of 51s (approximately 18s to 24s is due to kd-tree sorting overhead of $256^3$ (16 million) particles per processor).

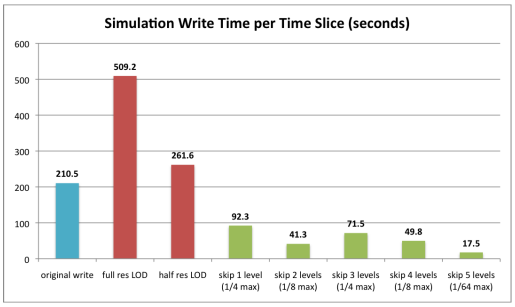


**Figure 11:** *Time taken to write sample and write the $MC^3$ particle data* in-situ *per time slice. The left-most bar is the original write time. The other bars are LOD storage with particle duplication.*

Actual simulation time savings comes from writing less data, which can be seen in Figure 12. Beginning with the 4th bar from the left in Figure 12, we save 118 seconds (2 minutes) per time slice. In the case where we are only writing ~1.5% of the original simulation data, we save 192s (3.2 minutes) per time slice. The time taken is nearly correlated by the amount of data written. Though, "skip 4 levels" takes 8 seconds longer than "skip 2 levels," even though the former stores less data. This is probably due to system noise and utilization, since Cerrillos is an actively used supercomputer and the Panasas file system is shared across several other supercomputers.

In Figure 13, we can see the amount of additional time slices that can be stored to increase time fidelity. If we sample the data at quarter resolution or less (assuming particle duplication), we can expect to be able to add approximately one to three additional time slices. This allows the cosmologists to make a trade-off between temporal and spatial fi-

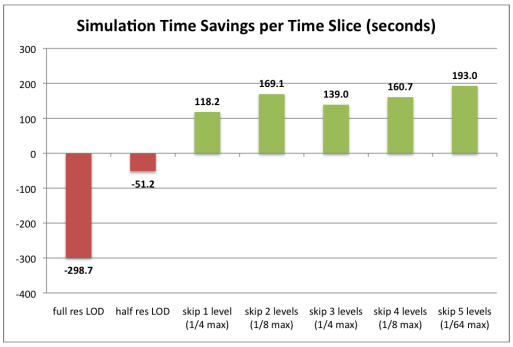


**Figure 12:** *Simulation time savings per time slice in different LOD configurations.*

delity using the same I/O time, which does not preclude writing out full resolution slices. For example, if the simulation normally writes 100 full resolution time steps, we can trade half of those for a sampled version (for example, every other full resolution time slice) to add between 100 to 200 additional time slices or potentially many more if using smaller resolutions. The remaining 50 could then be converted to 20 full resolution LOD time slices. This would create a a mix of full resolution LOD data and low resolution LOD data, with 200 to 300 time slices, in the same I/O time as before.

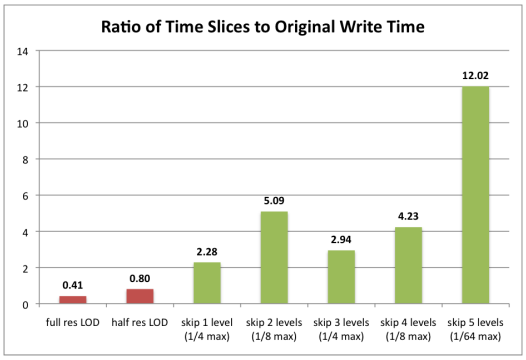


**Figure 13:** *Ratio of of our LOD write method time compared to the original write method time.*

Finally, we measure the performance of I/O read times for our LOD particle organization for different sample (block) sizes in Figure 14. The peak read rate from the disk we used is approximately 110 MB/s. In Section 4.2.1, we described a method for partially using blocks for intermediate LOD levels. As we can see, the larger block size (approximately 262 thousand samples or 8MB per block) has the best performance for full and partial I/O reads.

## 9. Conclusion

We have presented a method for *in-situ* stratified random sampling and level-of-detail storage for the RoadRunner

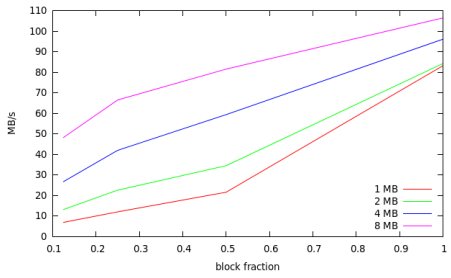


**Figure 14:** *Read block (sample size) I/O times, which vary based on partial read (striding) from a single mechanical disk. Peak throughput on a linear read is approximately 110MB/s on this disk.*

Universe MC$^3$ cosmological simulation for interactive post-analysis and visualization. We believe that this methodology opens a new *in-situ* venue to enable large-scale scientific analysis at extreme scales. This acts as a launching point for many different points of visualization research to support interactive post-analysis at exascale: enabling other large-scale simulations, enabling different data types (regular and irregular grids) and sampling methods, integration with compression [BR09, EGM05, LI06] and quantization methods [CMNR07, LHJ99, WGLS05], utilization of approximation error in analysis operations and the propagation of error, integration into existing *in-situ* and co-processing frameworks and/or using I/O function hooks, acquiring additional statistical and summary information of the original data, and using sampling and dimensionality reduction on data axes rather than only the spatial axes.

In particular for MC$^3$, we need to investigate measuring temporal error and ensuring temporal sample continuity. This would be similar to LOD visual continuity, because halo tracking and merging over time are an important research topic for the cosmologists. Also, we need to verify the applicability of our sampling for $n$-point correlation functions. Secondly, outlier information and visualization is important for verification and debugging of simulation codes [AHP*10]. We would like to look into sampling the data axes (variables) and biasing the sampling towards saving outlier data. Finally, we would like to look into using compact summary information to generate particle data at analysis time to increase the accuracy of halo finding for low resolution data.

### 9.1. Acknowledgments

This work was supported by the Department of Energy (DOE) National Nuclear Security Administration (NNSA) Advanced Simulation and Computing (ASC) Computational Systems and Software Environment (CSSE), DOE Office of Science (OSC) Advanced Scientific Computing Research (ASCR), and Los Alamos National Laboratory (LANL) Laboratory Directed Research and Development (LDRD).

## References


[AHL*10] AHRENS J., HENDRICKSON B., LONG G., MILLER S., ROSS R., WILLIAMS D.: *Data Intensive Science in the Department of Energy*. Tech. Rep. LA-UR-10-07088, Los Alamos National Laboratory, Oct. 2010. 2, 6

[AHP*10] AHRENS J., HEITMANN K., PETERSEN M., WOODRING J., WILLIAMS S., FASEL P., AHRENS C., HSU C., GEVECI B.: Verifying scientific simulations via comparative and quantitative visualization. *IEEE Computer Graphics and Applications 30*, 6 (2010), 16–28. 1, 6, 9

[AWD*09] AHRENS J. P., WOODRING J., DEMARLE D. E., PATCHETT J., MALTRUD M.: Interactive remote large-scale data visualization via prioritized multi-resolution streaming. In *UltraVis '09 Proceedings of the 2009 Workshop on Ultrascale Visualization* (Portland, Oregon, 2009), pp. 1–10. 2, 3, 6

[BR09] BURTSCHER M., RATANAWORABHAN P.: pFPC: a parallel compressor for floating-point data. In *Data Compression Conference, 2009. DCC '09.* (2009), pp. 43–52. 2, 9

[CJ10] CHEN M., JAENICKE H.: An information-theoretic framework for visualization. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1206–1215. 2, 6

[CMNR07] CLYNE J., MININNI P., NORTON A., RAST M.: Interactive desktop analysis of high resolution simulations: Application to turbulent plume dynamics and current sheet formation. *New Journal of Physics 9*, 8 (2007), 301–301. 2, 3, 9

[Coc77] COCHRAN W.: *Sampling techniques*, 3rd ed. Wiley, New York, 1977. 2, 3, 6

[CPA*10] CHILDS H., PUGMIRE D., AHERN S., WHITLOCK B., HOWISON M., PRABHAT, WEBER G., BETHEL E. W.: Extreme scaling of production visualization software on diverse architectures. *Computer Graphics and Applications, IEEE 30*, 3 (2010), 22–31. 2

[Dee98] DEERING M. F.: The limits of human vision. In *2nd International Immersive Projection Technology Workshop* (1998). 2

[EGM05] ELLSWORTH D., GREEN B., MORAN P.: Interactive terascale particle visualization. In *Visualization, 2004. IEEE* (2005), pp. 353–360. 2, 9

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run – scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1251–1258. 2, 3, 6

[HE03] HOPF M., ERTL T.: Hierarchical splatting of scattered data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 57. 2, 3, 6

[HPL*09] HABIB S., POPE A., LUKIĆ Z., DANIEL D., FASEL P., DESAI N., HEITMANN K., HSU C.-H., ANKENY L., MARK G., BHATTACHARYA S., AHRENS J.: Hybrid petacomputing meets cosmology: The roadrunner universe project. In *Journal of Physics: Conference Series* (2009), vol. 180, p. 012019. 2

[HPS09] HONG Y., PETERKA T., SHEN H.-W.: Histogram-based i/o optimization for visualizing large-scale data, November 2009. Slides from Ultrascale Visualization Workshop 2009. 5

[JR07] JOHNSON C., ROSS R.: *Visualization and Knowledge Discovery: Report from the DOE/ASCR Workshop on Visual Analysis and Data Exploration at Extreme Scale*. Tech. rep., Department of Energy Office of Science ASCR, Oct. 2007. 2, 6

[JS03] JOHNSON C., SANDERSON A.: A next step: Visualizing errors and uncertainty. *Computer Graphics and Applications, IEEE 23*, 5 (2003), 6–10. 2

[LHJ99] LAMAR E., HAMANN B., JOY K.: Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of the conference on Visualization'99: celebrating ten years* (1999), pp. 355–361. 2, 9

[LHJ03] LAMAR E., HAMANN B., JOY K.: Efficient error calculation for multiresolution texture-base volume visualization. In *Hierarchical and geometrical methods in scientific visualization*, Farin G., Hamann B., Hagen H., (Eds.). Springer, Feb. 2003, pp. 51–62. 2

[LI06] LINDSTROM P., ISENBURG M.: Fast and efficient compression of Floating-Point data. *Visualization and Computer Graphics, IEEE Transactions on 12*, 5 (2006), 1245–1250. 2, 9

[Loh10] LOHR S.: *Sampling: Design and Analysis*, 2nd ed. Brooks/Cole, Boston, MA, 2010. 2, 3, 6

[MFMG10] MORELAND K., FABIAN N., MARION P., GEVECI B.: *Visualization on Supercomputing Platform Level II ASC Milestone (3537-1B) Results from Sandia*. Tech. Rep. SAND2010-6118, Sandia National Laboratory, Sept. 2010. 2

[pan] Panasas resource library. http://www.panasas.com/products/library.php#productinfo. 2, 8

[PF01] PASCUCCI V., FRANK R.: Global static indexing for real-time exploration of very large regular grids. In *Supercomputing, ACM/IEEE 2001 Conference* (2001), p. 45. 2, 3

[PKRJ10] POTTER K., KNISS J., RIESENFELD R., JOHNSON C.: Visualizing summary statistics and uncertainty. *Computer Graphics Forum 29*, 3 (2010), 823–832. 2

[SBH*10] SZALAY A. S., BELL G. C., HUANG H. H., TERZIS A., WHITE A.: Low-power Amdahl-balanced blades for data intensive computing. *ACM SIGOPS Operating Systems Review 44*, 1 (2010), 71. 2

[TCM10] TIKHONOVA A., CORREA C., MA K.: Visualization by proxy: A novel framework for deferred interaction with volume data. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1551–1559. 2

[TYR*06] TU T., YU H., RAMIREZ-GUZMAN L., ACOBO BIELAK J., GHATTAS O., MA K., O'HALLARON D. R.: From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *Supercomputing, ACM/IEEE 2006* (Tampa, Florida, 2006), p. 91. 2

[Vit87] VITTER J. S.: An efficient algorithm for sequential random sampling. *ACM Transactions on Mathematical Software 13*, 1 (1987), 58–67. 2

[WGLS05] WANG C., GAO J., LI L., SHEN H.: A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Volume Graphics, 2005. Fourth International Workshop on* (2005), pp. 11–223. 2, 3, 9

[WGS07] WANG C., GARCIA A., SHEN H.-W.: Interactive level-of-detail selection using image-based quality metric for large volume visualization. *IEEE Transactions on Visualization and Computer Graphics 13*, 1 (2007), 122–134. 2

[WHA*10] WOODRING J., HEITMANN K., AHRENS J., FASEL P., HSU C. H., HABIB S., POPE A.: Analyzing and visualizing cosmological simulations with ParaView. *arXiv preprint arXiv:1010.6128* (2010). 6

[YWG*10] YU H., WANG C., GROUT R., CHEN J., MA K.-L.: In situ visualization for large-scale combustion simulations. *Computer Graphics and Applications, IEEE 30*, 3 (2010), 45–57. 2

**In-situ Sampling of a Large-Scale Particle Simulation for Interactive Visualization and Analysis**

J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann

Los Alamos National Laboratory, 2011

# Sampling and Level-of-Detail Construction Algorithm Pseudo-code

What follows is the algorithm (pseudocode) to recursively create a single stratified random sample of parallel particle data via a kd-tree per processing element as an SPMD (MPI-model) program. This is not the full algorithm but the basis for the full algorithm.

```
N = ‘‘total number of particles’’
S = ‘‘sample size’’ < N

# get local particles, sample the data
# with a fraction of the number of particles, and
# then gather the samples across all processors
p = get-local-particles()
gather(kd-tree-random-sample(p, S * |p| / N, 0))

# per-processor sampling function using a kd-tree.
# part is the particles, ss is the sample size,
# and d is the current kd-tree depth
function kd-tree-random-sample(part, ss, d):
  # if the sample size is one, get a random sample
  # from the kd-tree leaf (block/stratum)
  if ss == 1:
    return one-random-sample(part)
  # else split the particle data into two halves
  # and sample on each half, joining the samples
  else:
    left, right = median-sort(part, axis(d))
    return append(kd-tree-sample(left, ss/2, d+1),
                  kd-tree-sample(right, ss/2, d+1))
```

In the following, we show the modification of the previous pseudocode to provide the full algorithm and support level-of-detail (LOD) storage of the particles. The main differences are that the LOD levels are written to disk as they are generated, the data can be written out in full resolution in the LOD structure, and that the LOD structure is built and completed in parallel across processors (assuming that each processor has a spatial block of particles, as is in the case with $MC^3$).

This code assumes particle duplication per level for fast linear reads, thus every block or level has all of its particles without having to skip on disk to read and gather particles across multiple levels. It is entirely possible to easily modify the algorithm to remove the particle duplication per level and write particles only once.

```
N = ‘‘total number of particles’’
S = ‘‘sample size’’ < N
# V is slightly different here in implementation
# V is a subset of integers
V = ‘‘set of LOD levels to store’’
F = ‘‘LOD level to store as full resolution’’ >= -1

# get local particles and sample the data
# generate the LOD tree per processor
p = get-local-particles()
sample = sample-and-lod(p, S, 0)

# reduce the LOD tree from the per processor LOD trees
for d from -1 to -log2(number-of-processors()) by -1:
  # sibling is the opposite processor in a kd-tree
  # ordering of processors in simulation space
  sibling = get-opposite-sibling-rank(d)
  if get-rank() > sibling:
    send(sibling, sample)
    break
  else:
    if d is in V:
      sample =
        sieve-and-store
          (append(sample, receive(sibling)), d)

# per-processor sampling and LOD construction
# part is the particles, ss is the sample size,
# and d is the current kd-tree depth
function sample-and-lod(part, ss, d):
  # write the full resolution data as an LOD level
  # if it is enabled (not less than 0)
  if d == F:
    write-to-storage(part)
  # if the sample size is one, acquire a random sample
  # from the kd-tree leaf (block/stratum)
  if ss == 1:
    return one-random-sample(part)
  # else split the particle data into two halves
  # and build the LOD tree on both halves
```

```
  else:
    left, right = median-sort(part, axis(d))
    if d < max(V):
      sample = append(sample-and-lod(left, ss, d+1),
                      sample-and-lod(right, ss, d+1))
    else:
      sample = append(sample-and-lod(left, ss/2, d+1),
                      sample-and-lod(right, ss/2, d+1))
    # if it an LOD level, filter the samples
    if d is in V:
      return sieve-and-store(sample, d)
    # else return the blocks (samples) to the parent
    else:
      return sample

# build a random sample to pass to a parent block
# in the LOD tree and store this block (sample).
# there is a subtle nuance in that the particles
# that go to the parent are not randomly sampled
# from the entire pool, but one is sampled from
# every k sized partitions; this is to ensure that
# the one random particle per stratum holds at
# every level in the hierarchy
function sieve-and-store(sample, d):
  # every k sequential particles in the list are
  # from kd-tree siblings: randomly pick one of the
  # k particles to go to the parent block where
  # k = 2^(last stored LOD level - this LOD level)
  k = 2^(get-last-stored-level(V, d) - d)
  for i from 0 to S by k:
    pick = random(k)
    for j from 0 to k by 1:
      if j == pick:
        parent = append(parent, sample[i+j])
      else:
        children = append(children, sample[i+j])
    # random permutation of the children
    # is to support intermediate level rendering
    # described in the paper
    write-to-storage
      (append(parent, random-permutation(children))
    return parent
```